BEWD – Collections and Loops

Matt Heath
Senior Engineer, Hailo

AGENDA

- Iteration - Loops
- Collections
  - Arrays
  - Hashes

ITERATION

- Repetition
- Repetition
- Repetition

ITERATION

TIMES ITERATOR

```
3.times do
  puts “going...”
end
puts gone


# going...
# going...
# going...
# gone
```

ITERATION

TIMES ITERATOR

```
3.times do |num|
  puts “going #{num}”
end
puts gone


# going 1
# going 2
# going 3
# gone
```

ITERATION

.UPTO

```
1.upto(3) do |num|
  puts “#{num}. going”
end



# 1. going
# 2. going
# 3. going
```

ITERATION

.DOWNTO

```
3.downto(1) do |guess|
  puts “You have #{guess} guesses left”
end



# You have 3 guesses left
# You have 2 guesses left
# You have 1 guesses left
```

ITERATION

CONDITIONAL LOOPS

```
5.downto(1) do |count|
  puts “Looping”
end


# Looping
# Looping
# Looping
# Looping
# Looping
```

ITERATION

CONDITIONAL LOOPS

```
count = 5
while count > 0
  puts “Looping”
  count -= 1
end


# Looping
# Looping
# Looping
# Looping
# Looping
```

ITERATION

CONDITIONAL LOOPS

```
count = 5
until count < 1
  puts “Looping”
  count -= 1
end


# Looping
# Looping
# Looping
# Looping
# Looping
```

ITERATION

CONDITIONAL LOOPS

```
count = 5
loop do
  break if count -=< 1
  puts “Looping”
  count -= 1
end


# Looping
# Looping
# Looping
# Looping
# Looping
# Looping
```


LAB

```
# Write a program that prints 99 bottles of beer on the wall.
# The song starts with
# 99 bottles of beer on the wall
# 99 bottles of beer!
# You take one down and pass it around,
# 98 bottles of beer on the wall!
#
# And ends with
# 1 bottle of beer on the wall
# 1 bottle of beer!
# You take one down and pass it around,
# No more bottles of beer on the wall :-(
```

```
def bottle_count(count)
  if count == 1
    "#{count} bottle"
  else
    "#{count} bottles"
  end
end

99.downto(2) do |count|
  puts "#{bottle_count(count)} of beer on the wall"
  puts "#{bottle_count(count)} of beer"
  puts "You take one down and pass it around,"
  puts "#{bottle_count(count - 1)} of beer on the wall!"
  puts
end

puts "1 bottle of beer on the wall"
puts "1 bottle of beer"
puts "You take one down and pass it around,"
puts "No more bottles of beer on the wall :-("
```

```
def pluralise(word, count)
  "#{count} #{word}#{'s' unless count == 1}"
end

def sing_bottles(count)
  pluralised_count = pluralise("bottle", count)
  puts "#{pluralised_count} of beer on the wall"
  puts "#{pluralised_count} of beer"
  puts "You take one down and pass it around,"
  puts "#{pluralise("bottle", count - 1)} of beer on the wall!\n "
    unless count == 1
end

99.downto(2) do |count|
  sing_bottles(count)
end

sing_bottles(1)
puts "No more bottles of beer on the wall :-("
puts
```

ITERATION

RECAP

- Iteration in programming allows us to keep our code DRY
- Loops are used to repeat lines of code
- Common or Ruby-esque loops are
  - .times
  - .upto
  - .downto
  - .each (we will see in a moment)

COLLECTIONS

WHAT ARE ARRAYS?

- A type of variable
- Store several pieces of data
- Like a box with sections
- Ordered

ARRAYS

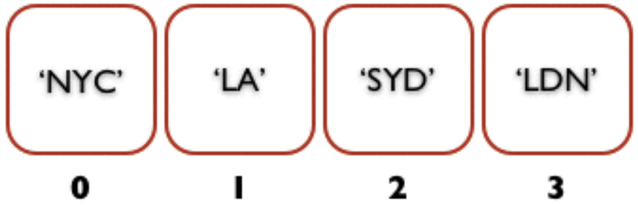
CREATING ARRAYS

```
# Blank arrays
my_array = Array.new
my_array = []

# A pre-populated array
my_array = ["NYC", "LA", "SYD", "LDN"]
```

ARRAYS

FIND BY INDEX

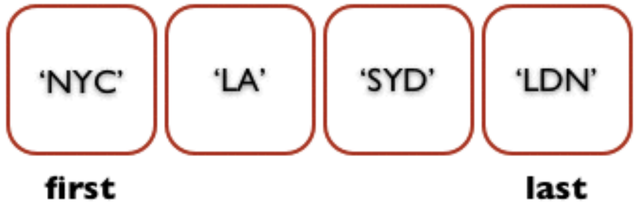



```
my_array = ["NYC", "LA", "SYD", "LDN"]

my_array[0]  #=> "NYC"
my_array[1]  #=> "LA"
my_array[-1] #=> "LDN"
```

ARRAYS

FIND BY POSITION




```
my_array = ["NYC", "LA", "SYD", "LDN"]

my_array.first  #=> "NYC"
my_array.last   #=> "LDN"
```

ARRAYS

FIND BY POSITION

```
# In rails...
# Will not work in IRB
my_array = ["NYC", "LA", "SYD", “LDN"]

my_array.second  #=> "LA"
my_array.third   #=> "SYD"
my_array.forth   #=> “LDN”

# known as the reddit
my_array.forty_two
```

ARRAYS

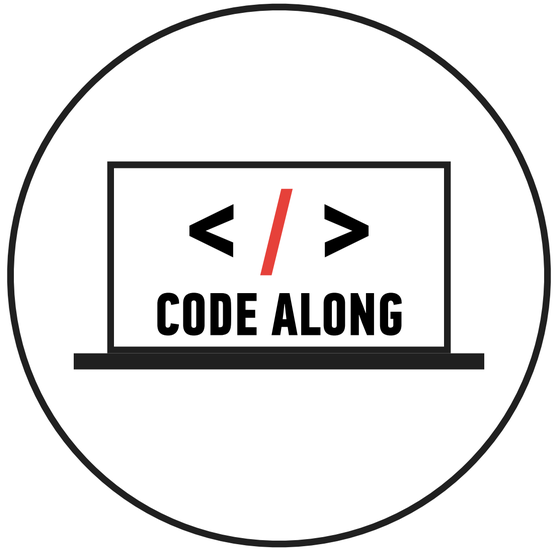
METHODS & ARRAYS

```
name = "Matt"
name.upcase #=> "MATT"

my_array = ["NYC", "LA", "SYD", "LDN"]
my_array.reverse
   #=> ["LDN", "SYD", "LA", "NYC"]
```

```
irb(main)> my_array.methods

=> [:inspect, :to_s, :to_a, :to_ary, :frozen?, :==, :eql?, :hash,
:[], :[]=, :at, :fetch, :first, :last, :concat, :<<, :push, :pop,
:shift, :unshift, :insert, :each, :each_index, :reverse_each, :length,
:size, :empty?, :find_index, :index, :rindex, :join, :reverse, :reverse!,
:rotate, :rotate!, :sort, :sort!, :sort_by!, :collect, :collect!, :map, :m
ap!, :select, :select!, :keep_if, :values_at, :delete, :delete_at, :delete
_if, :reject, :reject!, :zip, :transpose, :replace, :clear, :fill, :includ
e?, :<=>, :slice, :slice!, :assoc, :rassoc, :
+, :*, :-, :&, :|, :uniq, :uniq!, :compact, :compact!, :flatten, :flatten!
, :count, :shuffle!, :shuffle, :sample, :cycle, :permutation, :combination
, :repeated_permutation, :repeated_combination, :product, :take, :take_whi
le, :drop, :drop_while, :pack, :entries, :sort_by, :grep, :find, :detect,
:find_all, :flat_map, :collect_concat, :inject, :reduce, :partition, :grou
p_by, :all?, :any?, :one?, :none?, :min, :max, :minmax, :min_by, :max_by,
:minmax_by, :member?, :each_with_index, :each_entry, :each_slice, :each_co
ns, :each_with_object, :chunk, :slice_before, :nil?, :===, :=~, :!
~, :class, :singleton_class, :clone, :dup, :initialize_dup,
:initialize_clone, :taint, :tainted?, :untaint, :untrust, :untrusted?, :tr
ust, :freeze, :methods, :singleton_methods, :protected_methods, :private_m
ethods, :public_methods, :instance_variables, :instance_variable_get, :ins
tance_variable_set, :instance_variable_defined?, :instance_of?, :kind_of?,
:is_a?, :tap, :send, :public_send, :respond_to?, :respond_to_missing?, :ex
tend, :display, :method, :public_method, :define_singleton_method, :object
_id, :to_enum, :enum_for, :equal?, :!, :!
=, :instance_eval, :instance_exec, :__send__, :__id__]
```


< / >
CODE ALONG


LAB

ARRAYS

RECAP

- A collection of data
- Can search an array by index or position
- Arrays are objects so have methods we can call on them.

COLLECTIONS

WHAT ARE HASHES?

- Often referred to as dictionaries
- Each entry in a hash needs a **key** and a **value**
- If you access a hash at a specific key, it will return the value at that key

HASHES

FIND BY KEY

```
ga = {"NYC" => "New York City",
      "LA" => "Los Angeles”,
      "SYD" => "Sydney",
      "LDN" => "London"}

ga["LDN"] #=> "London"
ga["NYC"] #=> "New York City"
ga["SYD"] #=> "Sydney"
```

HASHES

SETTING VALUES

```
user_hash = {}
user_hash["name"] = "Matt"
user_hash["favourite_color"] = "Red"

user_hash
#=> {"name"=>"Matt",
    "favourite_color"=>"Red"}
```

SYMBOLS

SETTING VALUES

- A symbol is a special type of object in ruby, used extensively
- Starts with a colon

`:a`, `:b`, `:chunky`, and `:CHunKY_bACOn` are examples

- Like a lightweight string, usually used if you don't need to print to the screen
- Symbols are used because:
  - they are immutable and take less memory
  - they are easier to compare to other objects
  - they are cleaner in syntax

SYMBOLS

PRIMARILY USED AS HASH KEYS

```
ga = {}
ga = {:NYC => "New York City"}
ga[:LA] = "Los Angeles"

ga
#=> {:NYC => "New York City", :LA =>
"Los Angeles"}
```

HASHES

METHODS

```
user = {:user_name => "mattheath",
:email => "matt@mattheath.com"}

user.has_key? :email     #=> true
user.key? :email         #=> true
user.include? :email     #=> true

user.has_value? "mattheath" #=> true
(note: extremely inefficient!)
```

HASHES

ALTERNATIVE SYNTAX

```
user = {:user_name => "mattheath",
:email => "matt@mattheath.com"}

# becomes

user = {user_name: "mattheath",
email: "matt@mattheath.com"}

# a little bit more concise
# more closely matches JSON format
# considered an 'alternate' syntax
```

COLLECTIONS

ARRAYS OF HASHES

```
users = [
  {
    :user => “Matt Heath”,
    :role => “Instructor”
  },
  {
    :user => “CJ Ponti”,
    :role => “TA”
  }
]
```

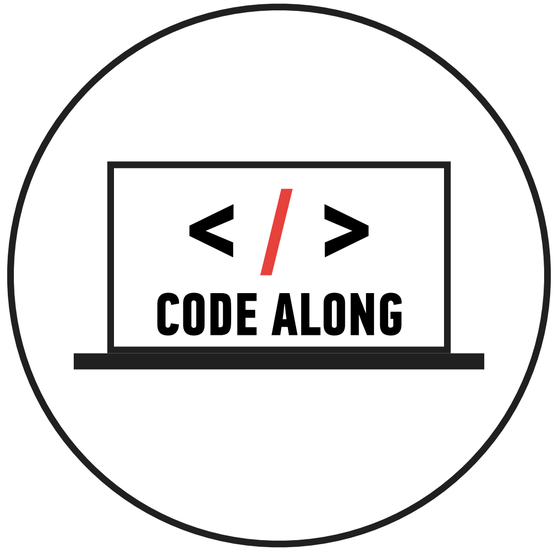
COLLECTIONS

ITERATING OVER COLLECTIONS WITH .EACH

```
courses = ["BEWD", "FEWD", "WDI"]

courses.each {|course| puts course}

>> "BEWD"
>> "FEWD"
>> "WDI"
```


< / >
CODE ALONG


LAB